PATENT ABSTRACTS OF JAPAN

(11)Publication number : 10-327154

(43)Date of publication of application : 08.12.1998

(51)Int.Cl. H04L 12/24
H04L 12/26
H04L 12/28

(21)Application number : 09-148459

(22)Date of filing : 23.05.1997

(71)Applicant : NEC CORP
NEC TELECOM SYST LTD

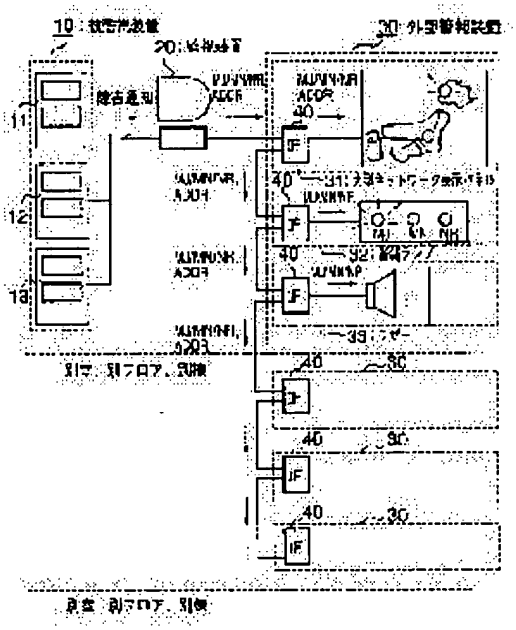
(72)Inventor : OSADA MASASHI
HORIE JUNICHI

(54) NETWORK MONITORING SYSTEM

(57)Abstract:

PROBLEM TO BE SOLVED: To facilitate extension or change by enabling daisy chain connection to an interface part on the side of each alarming device, receiving alarm output data at one alarming device and successively transmitting these data to the other alarming devices.

SOLUTION: When a monitoring device 20 receives a fault report from any device 10 to be monitored, concerning each fault, data showing the alarm level and required address of fault report are prepared and transmitted to an external alarming device 30 according to alarm output setting preset to an alarm output setting file. The alarming device 30 is equipped with a large network display panel 31 requiring the alarm level and the address, alarm lamp 32 and buzzer 33, which require only the alarm level, and issues an alarm while preparing data common for respective devices. Each IF part 40 has the same structure through the daisy chain connection and transmits the sent alarm level and address information as it is but at the lamp 32 or the buzzer 33, the information is ignored since no address is required.



EGAL STATUS

[Date of request for examination] 23.05.1997

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the

(19)日本国特許庁（ＪＰ）　(12)**公開特許公報（A）**　(11)特許出願公開番号

特開平10－327154

(43)公開日　平成10年(1998)12月８日

(51)Int.Cl.[6]	識別記号	FI		
Ｈ０４Ｌ　12/24		Ｈ０４Ｌ　11/08		
12/26			11/00	３１０Ａ
12/28				

審査請求　有　　請求項の数５　ＦＤ　（全 12 頁）

(21)出願番号　特願平9－148459

(22)出願日　平成９年(1997)５月23日

(71)出願人　000004237
日本電気株式会社
東京都港区芝五丁目７番１号

(71)出願人　000232106
日本電気テレコムシステム株式会社
神奈川県川崎市中原区小杉町１丁目403番地

(72)発明者　長田　正志
東京都港区芝五丁目７番１号　日本電気株式会社内

(72)発明者　堀江　準一
神奈川県川崎市中原区小杉町一丁目403番地　日本電気テレコムシステム株式会社内

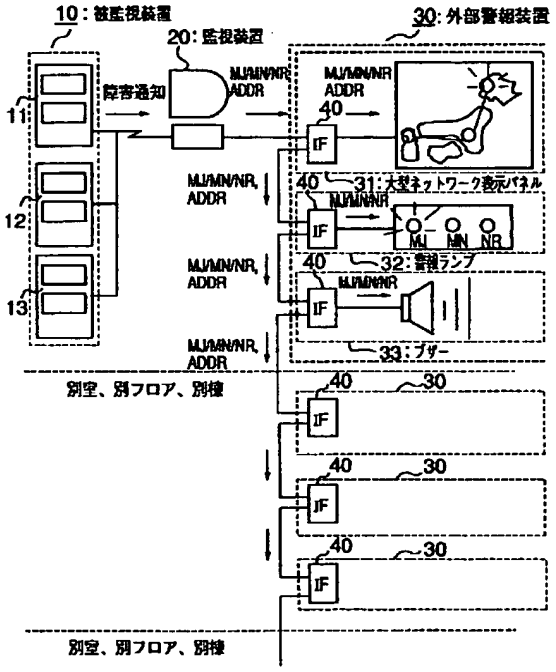
(74)代理人　弁理士　高橋　友二

(54)【発明の名称】　ネットワーク監視システム

(57)【要約】

【課題】　従来のネットワーク監視システムは警報装置の増設が容易でない、障害別に警報出力の制御や変更が行えない。何れかの警報装置の不一致の同期化が容易でない。

【解決手段】　監視装置２０から外部警報装置３０への出力を共通化し、各警報装置をＩＦ部４０でＲＳ２３２Ｃのディジーチェーン接続する。また監視装置２０に警報出力設定変更手段と警報出力一致手段を備え出力の制御，変更および一致化を図る。



1

【特許請求の範囲】

【請求項1】　ネットワークを構成する被監視装置からの障害通知（トラップ）を監視装置で受信し、複数種類の警報装置で構成された外部警報装置へ監視装置が警報出力データを送信し、各警報装置がこの警報出力データに基づいて警報を発するネットワーク監視システムにおいて、

各警報装置への警報出力データを共通化し、監視装置と各警報装置との間で共通のインタフェースを使用する手段、

各警報装置側のインタフェース部にディジーチェーン接続が可能なインタフェース部を使用し、前記監視装置が送信する前記警報出力データを1つの警報装置のインタフェース部で受信し、これをディジーチェーンで他の警報装置のインタフェース部に順次送信する手段、

を備えたことを特徴とするネットワーク監視システム。

【請求項2】　ネットワークを構成する被監視装置からの障害通知（トラップ）を監視装置で受信し、複数種類の警報装置で構成された複数の外部警報装置へ監視装置が警報出力データを送信し、各外部警報装置の各警報装置がこの警報出力データに基づいて警報を発するネットワーク監視システムにおいて、

各警報装置への警報出力データを共通化し、監視装置と各外部警報装置との間で共通のインタフェースを使用する手段、

各外部警報装置側のインタフェース部にディジーチェーン接続が可能なインタフェース部を使用し、前記監視装置が送信する前記警報出力データを1つの外部警報装置のインタフェース部で受信し、これをディジーチェーンで他の外部警報装置のインタフェース部に順次送信する手段、

を備えたことを特徴とするネットワーク監視システム。

【請求項3】　前記監視装置に記憶手段を備え、

前記記憶手段には、前記被監視装置からの障害通知を格納する障害発生状態ファイルと、前記障害通知に対し警報出力を行うか否か，警報出力を行う場合にその警報レベル，特定の警報装置に必要なアドレスを設定する警報出力設定ファイルと、前記障害発生ファイルの内容と前記警報出力設定ファイルの内容とから作成される前記警報出力データを逐次記憶しておく警報出力状態ファイルを設けたことを特徴とする請求項1又は請求項2記載のネットワーク監視システム。

【請求項4】　前記監視装置に対話型入出力装置を備え、

前記対話型入出力装置を用いて前記警報出力設定ファイルの内容を設定，変更する手段を備えたことを特徴とする請求項3記載のネットワーク監視システム。

【請求項5】　前記対話型入出力装置を用いて前記監視装置に前記警報状態ファイルに記憶した内容をすべて出力させて前記監視装置と前記各警報装置との同期化を図る手段を備えたことを特徴とする請求項3記載のネットワーク監視システム。

2

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明はネットワーク監視システム、さらに詳しくは監視装置が被監視装置からの障害通知を受けて複数の外部警報装置へ警報を出力して障害を知らせるネットワーク監視システムに関する。

【0002】

【従来の技術】ネットワーク監視システムは、監視対象であるネットを構成する電子交換機などの被監視装置からメッセージを受信し、そのメッセージをネットの特定の状態を表す情報を出力する。すなわち監視装置が被監視装置からの障害通知を受けて外部警報装置へ警報を出力するように構成されており、警報表示は一般的にメッセージクラスに基づいて行われる。

【0003】図7（A）は、従来のこの種のネットワーク監視システムを説明するための図であり、71は被監視装置、72は監視装置、73は外部警報装置である。また図7（B）は従来の監視装置72内に設けられている監視制御部である。外部警報装置73は、大型ネットワーク表示パネルの他に警報ランプ，ブザー等の各種の監視装置で構成されており、監視装置72は、何れかの被監視装置から障害通知を受信すると、警報装置選択処理部721が動作してその障害通知の内容に応じて大型表示パネル，警報ランプ，ブザーを単数または複数選択して警報を行う。またメッセージクラス、すなわち重要度であるMJ（メジャー），MN（マイナー），NR（ノーマル）を決定する。なお、MJは重大な障害の発生，MNは小さな障害の発生，NRは正常稼働状態を意味する。

【0004】外部警報装置の大型表示パネル，警報ランプ，ブザーは、それぞれインターフェースが異なるので、警報装置選択処理部721はそれぞれの送信データ作成／通信制御部722～724を動作させて各警報装置ごとの個別プロトコルの送信データを作成して各警報装置へ送信する必要がある。

【0005】

【発明が解決しようとする課題】従来のネットワーク監視システムは以上のように構成され動作するが、監視装置はワークステーションやパソコンを用いたものが多く、ポート数に制限があるため接続できる外部警報装置の数が制限されてしまう。従って外部警報装置を各室に設置などが困難になり、オペレータは常に外部警報装置の前で監視している必要がある。

【0006】また各警報装置の接続インタフェースがそれぞれ異なるため個別のプロトコルを用意する必要があり、警報装置の増設や変更においては、監視装置のプログラムを変更する必要がある。

【0007】また従来のネットワーク監視システムで

は、上述のメッセージクラスMJ，MN，NRが一律に定められオペレータによるカスタマイズが行えないため、例えば増設工事中や故障修理待ちの期間等、オペレータが既に承知している場合にも警報が出力されてしまい、これを防止するには警報装置の電源をOFFする必要があるが、警報装置をOFFすると他の被監視装置からの障害通知も警報できなくなるという不都合が生じる。

【0008】さらに監視装置と各警報装置とは、電源ON／OFFが独立しているため監視装置と各警報装置の警報発出状態に不一致が生じる可能性があり、不一致が生じた場合容易に同期化できない。すなわち従来のネットワーク監視システムの監視装置は、被監視装置からの障害通知を受けて各警報装置へ一度警報出力データを送出するだけなので、警報装置の何れかの電源がOFFしていた場合、当該警報装置の警報状態が監視装置と不一致になる等の問題点があった。

【0009】本発明はかかる問題点を解決するためになされたものであり、監視装置の出力ポートに制限を受けることなく必要な台数の外部警報装置を設置でき、その増設も容易に行え、特定の被監視装置からの警報を制御するなどオペレータによるカスタマイズが可能で、さらに監視装置と各警報装置の警報状態の不一致を容易に同期化できるネットワーク監視システムを提供することを目的としている。

【0010】

【課題を解決するための手段】本発明のネットワーク監視システムは、ネットワークを構成する被監視装置からの障害通知（トラップ）を監視装置で受信し、複数種類の警報装置で構成された外部警報装置へ監視装置が警報出力データを送信し、各警報装置がこの警報出力データに基づいて警報を発するネットワーク監視システムにおいて、各警報装置への警報出力データを共通化し、監視装置と各警報装置との間で共通のインタフェースを使用する手段、各警報装置側のインタフェース部にディジーチェーン接続が可能なインタフェース部を使用し、前記監視装置が送信する前記警報出力データを１つの警報装置のインタフェース部で受信し、これをディジーチェーンで他の警報装置のインタフェース部に順次送信する手段を備えたことを特徴とする。従って外部警報装置に設ける各警報装置の増設や変更が容易に行えるようになり、個別のプロトコルを用意する必要がなくなる。

【0011】また、ネットワークを構成する被監視装置からの障害通知（トラップ）を監視装置で受信し、複数種類の警報装置で構成された複数の外部警報装置へ監視装置が警報出力データを送信し、各外部警報装置の各警報装置がこの警報出力データに基づいて警報を発するネットワーク監視システムにおいて、各警報装置への警報出力データを共通化し、監視装置と各外部警報装置との間で共通のインタフェースを使用する手段、各外部警報装置側のインタフェース部にディジーチェーン接続が可能なインタフェース部を使用し、前記監視装置が送信する前記警報出力データを１つの外部警報装置のインタフェース部で受信し、これをディジーチェーンで他の外部警報装置のインタフェース部に順次送信する手段を備えたことを特徴とする。従って外部警報装置の増設が容易に行えるようになり、また各警報装置個別のプロトコルを用意する必要がなくなる。

【0012】また前記監視装置に記憶手段を備え、前記記憶手段には、前記被監視装置からの障害通知を格納する障害発生状態ファイルと、前記障害通知に対し警報出力を行うか否か，警報出力を行う場合にその警報レベル，特定の警報装置に必要なアドレスを設定する警報出力設定ファイルと、前記障害発生ファイルの内容と前記警報出力設定ファイルの内容とから作成される前記警報出力データを逐次記憶しておく警報出力状態ファイルを設けたことを特徴とする。

【0013】また前記監視装置に対話型入出力装置を備え、前記対話型入出力装置を用いて前記警報出力設定ファイルの内容を設定，変更する手段を備えたことを特徴とする。従ってオペレータによるカスタマイズが可能となり、増設工事中の期間に警報が発せられ続ける等の不都合を解消できる。

【0014】さらに、前記対話型入出力装置を用いて前記監視装置に前記警報状態ファイルに記憶した内容をすべて出力させて前記監視装置と前記各警報装置との同期化を図る手段を備えたことを特徴とする。従って警報装置の電源がOFFされていた場合に生じる不一致を容易に解消できる。

【0015】

【発明の実施の形態】以下、本発明の実施形態を図面を用いて説明する。図１は、本発明のネットワーク監視システムの装置構成の一実施形態を示すブロック図であり、図において、１０は被監視装置、２０は監視装置、３０は外部警報装置である。接続される被監視装置１０は例えば１１～１３の複数台の被監視装置が存在する。また監視装置２０には、記憶装置，対話型入出力装置が設けられ（共に図示せず）、記憶装置には、図２に示すように、障害発生状態ファイル２０１，警報出力設定ファイル２０２，警報出力状態ファイル２０３が格納されている。また外部警報装置３０は、例えば大型ネットワーク表示パネル３１，警報ランプ３２，ブザー３３等の各警報装置で構成され、各警報装置には、それぞれにインタフェース（ＩＦ）部４０が設けられている。図３は、ＩＦ部４０の構成を示すブロック図であり、４０１はバイパス用リレー回路、４０２は各ＩＦ部４０のデータの送信および受信を、それぞれ送信，受信の単位でＡＮＤするゲート（実際にはソフトウェアで構成されたゲート）であり、これにより複数台のＩＦ部４０をディジーチェーン接続できるように構成されている。

5

【0016】監視装置20が何れかの被監視装置10から障害通知（トラップ）を受信すると、個々のトラップに対し、警報出力設定ファイル（図2の202）に予め設定した警報出力設定に従い、障害通知の警報レベルMJ（メジャー），MN（マイナー），NR（ノーマル）および必要なアドレス（例えばパネルのどの部分に警報表示を行うかのアドレス）のデータを作成し、外部警報装置30へ送信する。外部警報装置30には、警報レベルとアドレスが必要な大型ネットワーク表示パネル31，警報レベルのみを必要とする警報ランプ32，ブザー33の各警報装置があるが、本実施形態では各警報装置に共通なデータを作成して警報を発出する。各IF部40は同じ構成なので、警報ランプ32，ブザー33に対しても、送られてきた警報レベルとアドレス情報をそのまま送信するが、警報ランプ32やブザー33ではアドレスを必要としないので無視する。

【0017】また図3に示すように、監視装置20と各IF部40とはRS－232Cコネクタとケーブルとを用いて接続されており、IF部40に設けられたバイパス用リレー回路401は、IF部40の電源OFFを検出するとバイパス回路を構成するようにリレースイッチ（SW1，SW2）が動作する。またIF部40では、ゲート402により各IF部40の送信および受信データが送信，受信の単位でANDされ、図1に示すように複数のIF部40がディジーチェーン接続される。

【0018】次に図2の監視装置20の記憶装置内に格納される各ファイルの内容について説明する。障害発生状態ファイル201は、トラップで通知される全ての障害について、障害発生中か否かを格納する。障害発生状態はトラップで通知された障害を記憶する。警報出力設定ファイル202は、障害通知別に、オペレータが対話型入出力装置を用いて行った、警報出力のON／OFF（警報出力する／しない）の設定、警報レベル（重要度）MJ／MN／NRの設定、大型表示パネルに表示する際のアドレス情報の設定およびコメント入力が記憶される。これらの設定は、本実施形態ではオペレータが対話型入出力装置のキーボードやマウスを用いて希望通りに設定，変更できるように構成されている。警報出力状態ファイル203は、障害発生ファイル201の内容と警報出力設定ファイル202の内容とから監視装置20で作成され出力される警報出力データを逐次記憶しておくファイルで、全てのアドレス部と現在発出中の警報レベルの警報出力状態が記憶される。

【0019】図4（A）は監視装置20が障害通知（トラップ）を受信した時の動作を示すフローチャートである。監視装置20はトラップを受信すると、トラップで通知された障害に基づいて障害発生状態ファイル201の内容を更新する。次に監視装置20は、警報出力設定ファイル202を参照し、警報出力設定がONであれば警報レベルに従いMJ／MN／NRを警報出力状態ファイル203に記録更新し、外部警報装置30に警報レベル（MJ／MN／NR）とアドレス情報を出力する。また警報出力設定ファイルの警報出力設定がOFFの場合には、警報出力状態ファイル203の更新は行わず、外部警報装置30への出力も行わない。

6

【0020】図4（B）は、警報出力設定変更処理および警報同期化処理を示すフローチャートである。警報出力設定変更処理では、オペレータが対話型入出力装置のキーボードやマウスを用いて、警報出力設定ファイル202の内容を変更し、続いて警報出力状態ファイル203を対話型入出力装置に読み出し、変更した警報出力設定に従い警報出力状態ファイル203の内容を更新し、このファイル203からの全ての情報を外部警報装置30へ送信する。

【0021】次に警報同期化処理について説明する。警報同期化処理は、監視装置20側と外部警報装置30側との両方から行うことが可能であるが、監視装置20側から行う場合には、対話型入出力装置のキーボードやマウスを用いて監視装置20へ警報同期化処理要求コマンドを出力する。監視装置20はこのコマンドを受信して警報出力状態ファイル203にある全ての情報を外部警報装置30へ送信し、各警報装置の全ての表示パネル，ランプ，ブザーの状態を、監視装置20が持つ現在の状態に一致させる。すなわち電源がOFFされていて不一致になった警報装置を監視装置20が持つ現在の状態に同期化させる。また外部警報装置30側から監視装置20に対し警報同期化処理要求コマンドを出力する場合には、警報装置側は、RS232Cのマルチドロップのスレーブ側にあたるので、スレーブ側の同期要求の衝突を考慮した手順で（ポーリング方式，セレクティング方式などを用いて）送信する。

【0022】次に図1，図2を用いて本発明の実施例を説明する。例えば被監視装置10で「xxx基幹回線障害」が発生した場合、障害通知（トラップ）で監視装置20に「xxx基幹回線障害　発生」が通知される。監視装置20はこのトラップを受信すると、障害発生状態ファイル201に「xxx基幹回線障害　発生」を格納する。そして警報出力設定ファイル201を参照してxxx基幹回線障害に該当するレコードを検索する。

【0023】監視装置20は、xxx基幹回線障害に該当するレコードを見付けた場合、このレコードの出力，警報レベル，アドレスの設定に従い、外部警報装置30に出力する出力情報を警報状態出力ファイル203に記録する。警報出力設定ファイル202では、出力がON，警報レベルはMJ（メジャー），アドレスはxxxなので、外部警報装置30に出力する情報として、（MJ発生／MNなし／NRなし，xxx）というデータを作成して送信する。各外部警報装置30は、このデータ（MJ発生，xxx）を受信し、各々の警報装置で警報を表示，鳴動する。

【００２４】次に各警報装置の動作を説明する。大型ネットワーク表示パネル３１では、データ（ＭＪ発生，ｘｘｘ）を受信し、ｘｘｘのアドレスが示す回線をＭＪ色（例えば赤色）で表示する。また警報ランプ３２では、データ（ＭＪ発生，ｘｘｘ）を受信し、ＭＪ色（例えば赤色）のランプを点灯する（アドレスは必要ないので無視する）。さらにブザー３３では、データ（ＭＪ発生，ｘｘｘ）を受信し、ＭＪ（例えば最大音量で復旧するまで連続鳴動）でブザーを鳴動させる（アドレスは必要ないので無視する）。また別室（別フロア，別棟）においても、ＩＦ部４０間でディジーチェーン接続された同様の外部警報装置３０が設置されており、各警報装置がデータ（ＭＪ発生，ｘｘｘ）を受信できるので、同じように警報動作が行われる。

【００２５】次に工事中などの理由により、現在出力中のｘｘｘ基幹回線障害警報を停止／再開する操作について、図２，図４を用いて説明する。オペレータは対話型入出力装置のキーボードやマウスを用いて、図２の警報出力設定ファイル２０２のｘｘｘ基幹回線障害の出力をＯＮからＯＦＦに変更する。また必要に応じて現在工事中である旨のコメントを加える。監視装置２０はこの出力の設定変更に従い、障害発生状態ファイル２０１と警報出力設定ファイル２０２とから、警報出力状態ファイル２０３の当該アドレスの警報状態を更新し、外部警報装置３０に送信する。

【００２６】警報停止を復旧する場合は、警報出力状態ファイル２０３の当該アドレスの警報状態を単に復旧するだけでなく、例えばｂｂｂ装置故障（アドレスｂｂｂ）、ｃｃｃカード故障（アドレスｂｂｂ）のように、一つのアドレスで複数の障害状態を含んで警報表示していることがあるので、該当するアドレスでそれまでに発生していた障害状態を再度判定して、警報出力状態ファイル２０３を更新する必要がある。同様の方法で、警報の出力の設定変更のみならず、警報出力設定ファイル２０２の警報レベル，アドレスの設定変更が行える（警報出力変更のフローでは、簡略して全ての警報を再度出力することとしている）。

【００２７】最後に同期化処理について説明する。この場合は、警報出力状態ファイル２０３の内容の全てを出力し、各警報装置の全ての表示パネル，ランプ，ブザーの状態を監視装置が持つ現在の状態に一致させる。従来のネットワーク監視システムでは、送信した警報出力データを記憶する手段をもたないので、警報装置の何れかがそのデータ送信時に電源がＯＦＦされていた場合、監視装置２０と不一致になり、これを同期化するのが容易でないが、本実施形態では警報出力状態ファイル２０３の内容の全てを出力させることで容易に同期化できるようになる。

【００２８】図５は本発明の他の実施形態を示すブロック図である。図１に示す実施形態では、外部警報装置３０内にＩＦ部４０を設ける構成としているが、図５に示すようにＩＦ部４０を外部警報装置３０から独立させる構成としても良い。

【００２９】また図６は本発明の更に他の実施形態を示すブロック図である。図１に示す実施形態では、それぞれの警報装置にそれぞれＩＦ部４０を設ける構成としているが、ＩＦ部４０のコネクタ数を複数とすることで、各室（フロア，棟）に１台づつ設置される外部記憶装置３０にそれぞれ一つのＩＦ部４０を設ける構成としても良い。

【００３０】

【発明の効果】以上説明したように本発明のネットワーク監視システムは、各警報装置への警報出力データを共通化し、インタフェース部をディジーチェーン接続することとしたので、警報装置の増設や変更，さらに外部警報装置を複数室に設ける等のシステムの拡張が容易に行えるようになる。また監視装置は各警報装置毎に警報出力データを作成する必要がなくなる。また警報出力設定ファイルの内容を変更することで、オペレータが個別に警報装置を制御でき、これによって増設期間中に警報が発せられ続ける等の不都合を解消できる。さらに警報状態ファイルに記憶した内容をすべて出力させて監視装置と各警報装置との同期化を図る手段を備えることとしたので、警報装置の警報状態の不一致を容易に解消できる等の効果がある。

【図面の簡単な説明】

【図１】本発明の一実施形態を示すブロック図である。

【図２】本実施形態で記憶装置に格納されるファイルを示す図である。

【図３】本実施形態のＩＦ部の構成を示すブロック図である。

【図４】本実施形態の動作を示すフローチャートである。

【図５】本発明の他の実施形態を示すブロック図である。

【図６】本発明の更に他の実施形態を示すブロック図である。

【図７】従来のネットワーク監視システムを示す図である。

【符号の説明】

１０　被監視装置
２０　監視装置
３０　外部警報装置
３１　大型ネットワーク表示パネル
３２　警報ランプ
３３　ブザー
４０　インタフェース部
２０１　障害発生状態ファイル
２０２　警報出力設定ファイル
２０３　警報出力状態ファイル

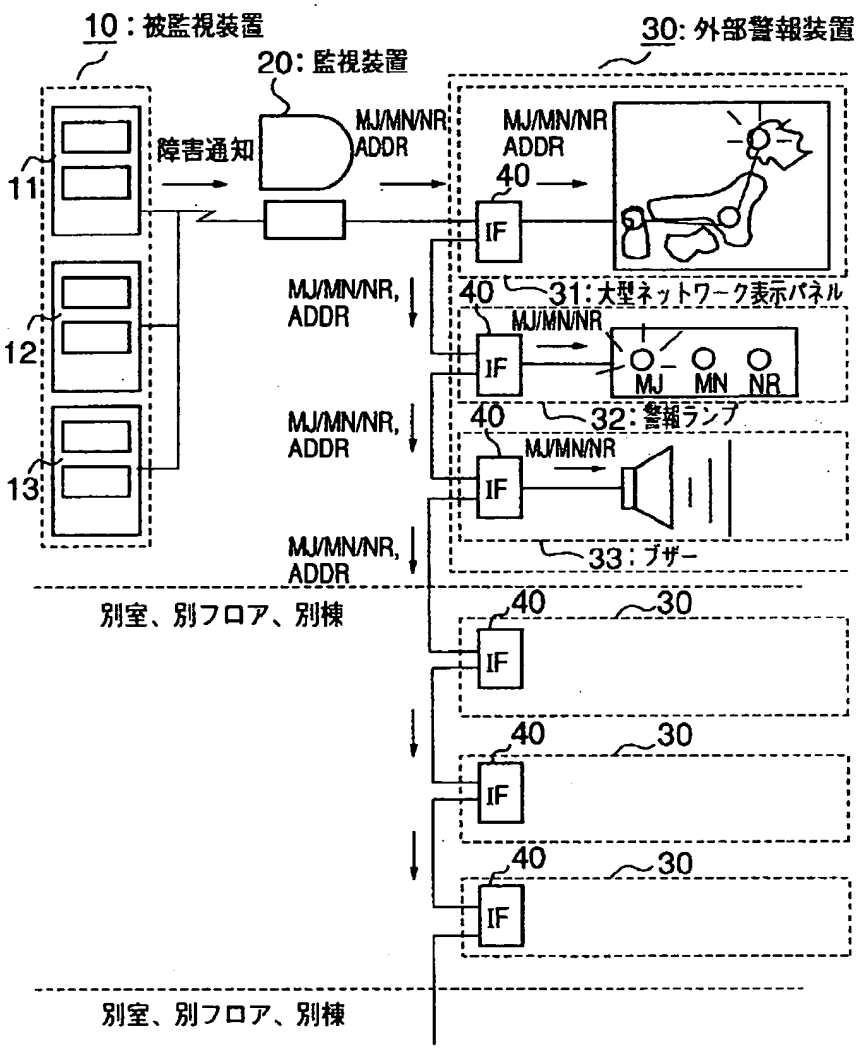
9

401　バイパス用リレー回路
402　ゲート
722　大型表示パネル用送信データ作成／通信制御部

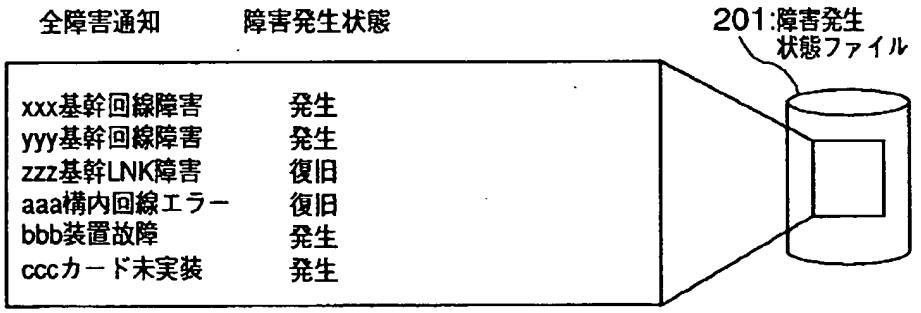
10

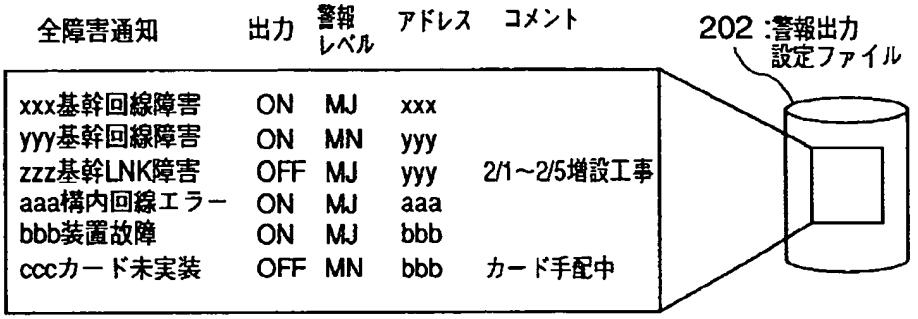
723　警報ランプ用送信データ作成／通信制御部
724　ブザー用送信データ作成／通信制御部

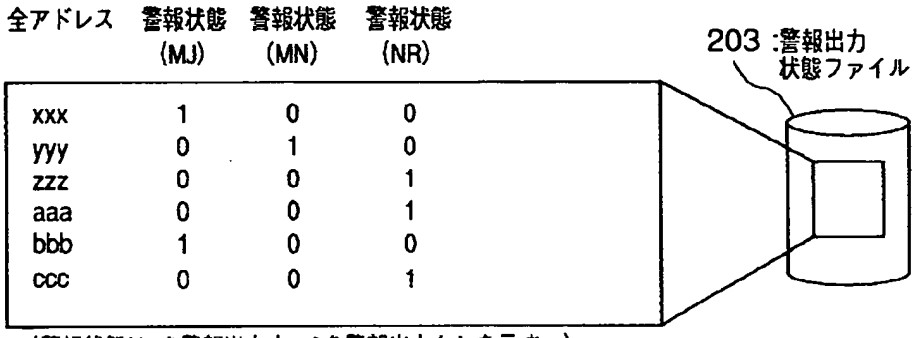
【図1】



【図2】

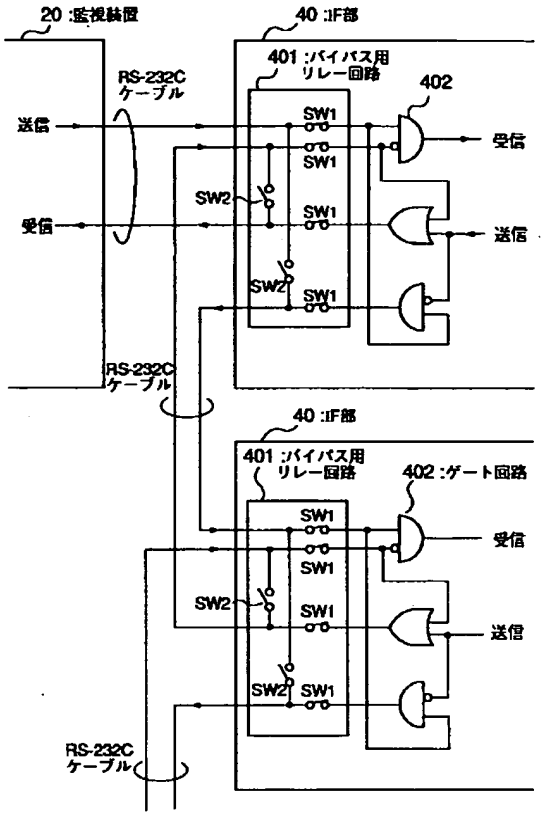






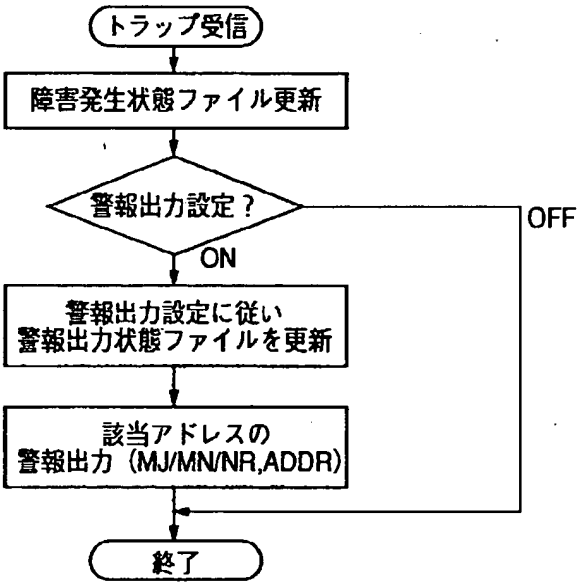
（警報状態は1を警報出力中、0を警報出力なしを示す。）

【図3】

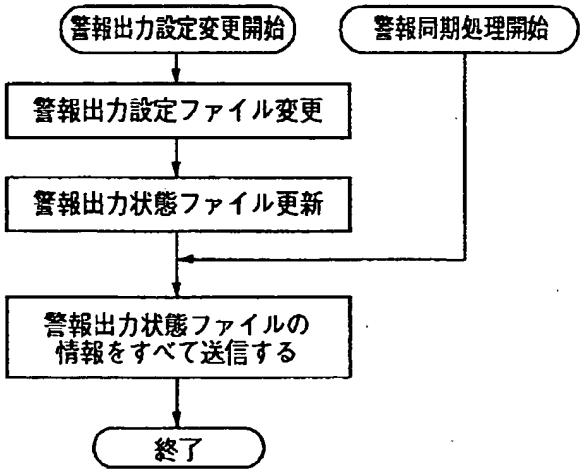

20 :監視装置
40 :IF部
401 :バイパス用
リレー回路
402
RS-232C
ケーブル
送信
受信
SW1
SW2
受信
送信
RS-232C
ケーブル
40 :IF部
401 :バイパス用
リレー回路
402 :ゲート回路
SW1
SW2
受信
送信
RS-232C
ケーブル

【図4】

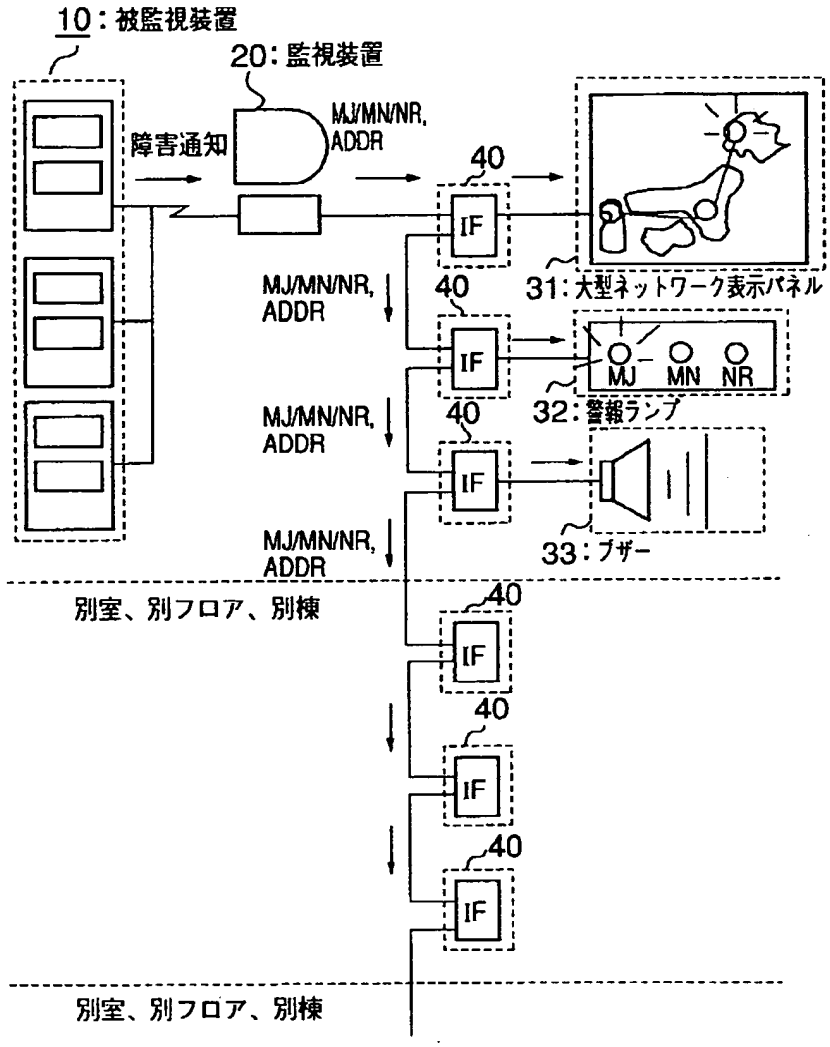
(A)


トラップ受信
障害発生状態ファイル更新
警報出力設定？
OFF
ON
警報出力設定に従い
警報出力状態ファイルを更新
該当アドレスの
警報出力（MJ/MN/NR,ADDR）
終了

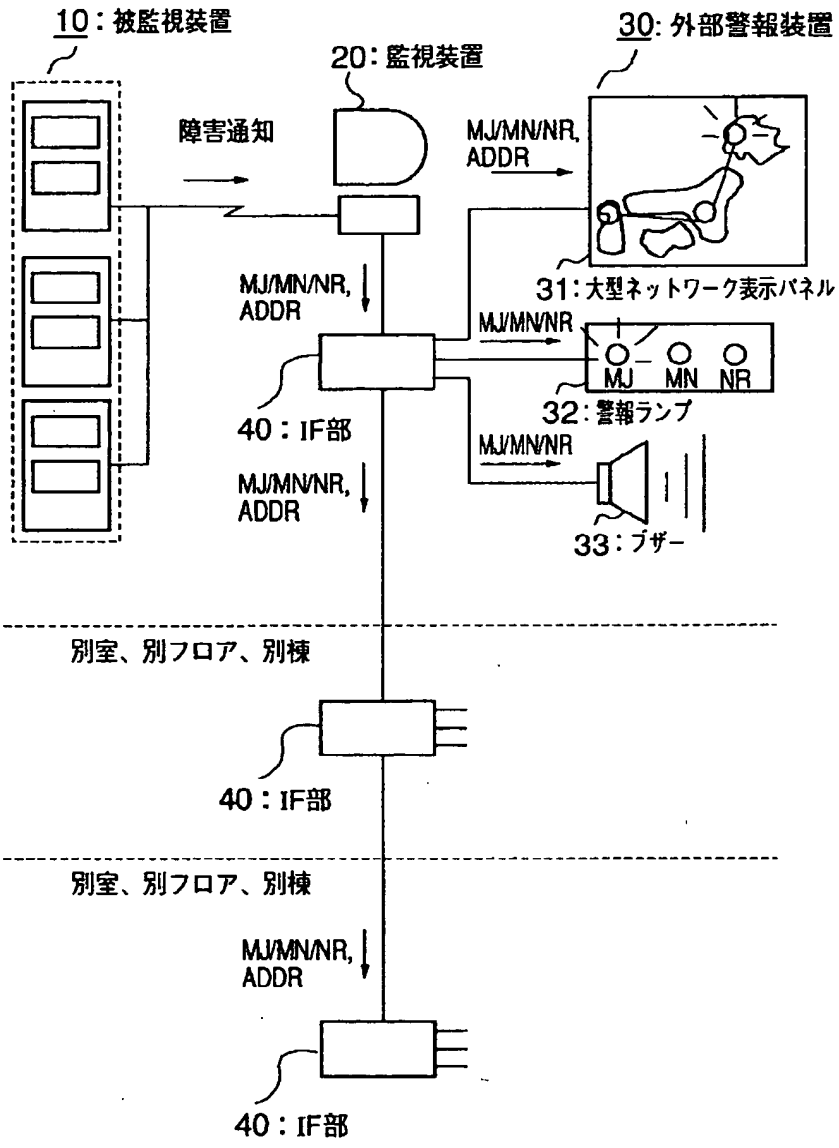
(B)


警報出力設定変更開始
警報同期処理開始
警報出力設定ファイル変更
警報出力状態ファイル更新
警報出力状態ファイルの
情報をすべて送信する
終了

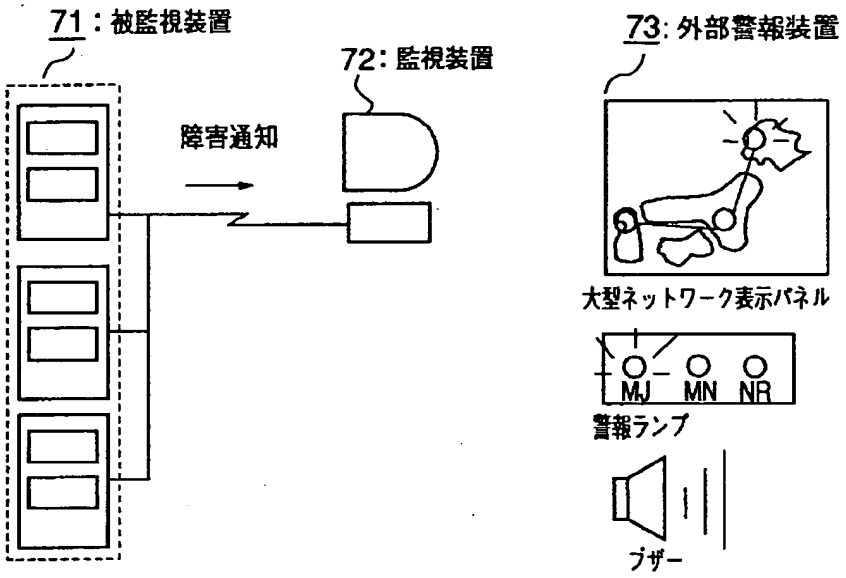
【図5】


10：被監視装置
20：監視装置
障害通知
MJ/MN/NR,
ADDR
40
IF
31：大型ネットワーク表示パネル
MJ/MN/NR,
ADDR
MJ
MN
NR
32：警報ランプ
33：ブザー
別室、別フロア、別棟
別室、別フロア、別棟

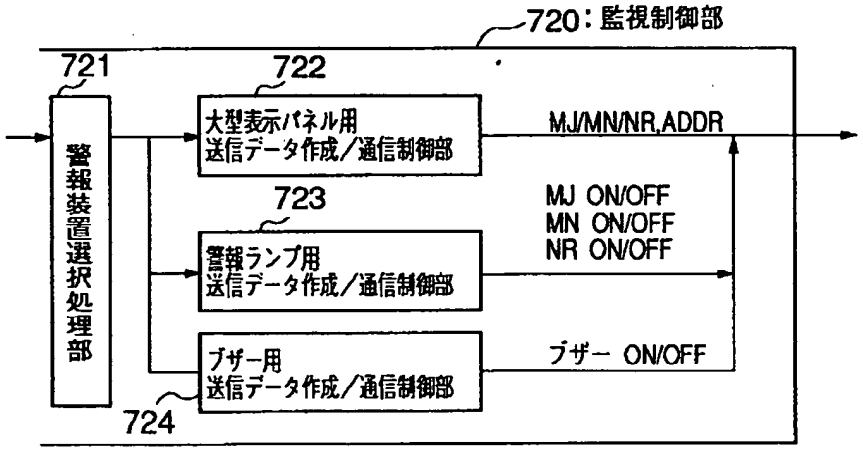
【図6】


10：被監視装置
20：監視装置
30：外部警報装置
障害通知
MJ/MN/NR,
ADDR
31：大型ネットワーク表示パネル
MJ/MN/NR
MJ
MN
NR
32：警報ランプ
40：IF部
33：ブザー
別室、別フロア、別棟
別室、別フロア、別棟

【図7】


71：被監視装置
72：監視装置
障害通知
73：外部警報装置
大型ネットワーク表示パネル
MJ
MN
NR
警報ランプ
ブザー

(A)


720：監視制御部
721
警報装置選択処理部
722
大型表示パネル用
送信データ作成／通信制御部
MJ/MN/NR,ADDR
723
警報ランプ用
送信データ作成／通信制御部
MJ ON/OFF
MN ON/OFF
NR ON/OFF
ブザー用
送信データ作成／通信制御部
ブザー ON/OFF
724

(B)

DETAILED DESCRIPTION

[Detailed Description of the Invention]
[0001]
[Field of the Invention] This invention relates to network monitoring system and the network monitoring system with which supervisory equipment outputs an alarm to two or more external alarms in response to the advice of a failure from supervisory equipment-ed, and tells a failure in more detail.
[0002]
[Description of the Prior Art] Network monitoring system receives a message from supervisory equipment-ed, such as the electronic automatic exchange which constitutes the network which is an object for a monitor, and outputs the information which expresses the specific condition of a network for the message. That is, it is constituted so that supervisory equipment may output an alarm to an external alarm in response to the advice of a failure from supervisory equipment-ed, and generally an alarm display is performed based on a message class.
[0003] Drawing 7 (A) is drawing for explaining this conventional kind of network monitoring system, and, as for supervisory equipment-ed and 72, 71 is [supervisory equipment and 73] external alarms. Moreover, drawing 7 (B) is the supervisory-control section prepared in conventional supervisory equipment 72. if the external alarm 73 consists of various kinds of supervisory equipment other than a large-sized network display panel, such as an alarm lamp and a buzzer, and supervisory equipment 72 receives advice of a failure from which supervisory equipment-ed -- the alarm selection processing section 721 -- operating -- the content of the advice of a failure -- responding -- a large-sized display panel, an alarm lamp, and a buzzer -- an unit -- or a multiple selection is made and an alarm is performed. Moreover, MJ (measure), MN (minor), and NR (Normal) which are a message class, i.e., significance, are determined. In addition, generating of a failure with serious MJ, generating of the failure that MN is small, and NR mean normal operating status.
[0004] Since, as for the large-sized display panel of an external alarm, an alarm lamp, and a buzzer, interfaces differ, respectively, the alarm selection processing section 721 needs to operate each transmit data creation / communications control section 722-724, needs to create the transmit data of the individual protocol for every alarm, and needs to transmit to each alarm.
[0005]
[Problem(s) to be Solved by the Invention] Although the conventional network monitoring system is constituted as mentioned above and it operates, supervisory equipment will have many which used the workstation and the personal computer, and the number of external alarms connectable since the number of ports has a limit will be restricted. Therefore, installation etc. needs to become difficult at each ** and the operator needs to always be supervising the external alarm in front of an external alarm.] Reason
[0006] Moreover, since the connection interfaces of each alarm differ, respectively, it is necessary to prepare the protocol according to individual, and in duplication and modification of an alarm, it is necessary to change the program of supervisory equipment.
[0007] Moreover, in the conventional network monitoring system, since the above-

mentioned message classes MJ, MN, and NR are defined uniformly and customize by the operator cannot be performed, the period of the waiting under duplication work and for failure correction etc. needs to turn off the power source of an alarm for an alarm to be outputted and prevent this, also when the operator already consents, but if an alarm is turned off, the inconvenience of stopping being able to carry out the alarm also of the advice of a failure from other supervisory equipment-ed will arise.

[0008] Furthermore, supervisory equipment and each alarm cannot be easily synchronized, when an inequality may arise in the alarm generation condition of supervisory equipment and each alarm and an inequality arises, since power-source ON/OFF is independently. That is, since the supervisory equipment of the conventional network monitoring system only sent out alarm-output data to each alarm once in response to the advice of a failure from supervisory equipment-ed, when which power source of an alarm turned off, it had troubles, like the alarm state of the alarm concerned becomes supervisory equipment and an inequality.

[0009] It is made in order that this invention may solve this trouble, the external alarm of the required number can be installed, without receiving a limit in the output port of supervisory equipment, the duplication can also be performed easily, and customize by the operator, such as controlling the alarm from specific supervisory equipment-ed, is possible, and it is aimed at offering the network monitoring system which can synchronize easily the inequality of the alarm state of supervisory equipment and each alarm further.

[0010]

[Means for Solving the Problem] The network monitoring system of this invention receives the advice of a failure from the supervisory equipment-ed which constitutes a network (trap) with supervisory equipment. In the network monitoring system with which supervisory equipment transmits alarm-output data to the external alarm which consisted of two or more kinds of alarms, and each alarm emits an alarm based on this alarm-output data A means to communalize the alarm-output data to each alarm, and to use a common interface between supervisory equipment and each alarm, The interface section in which daisy chain connection is possible is used for the interface section by the side of each alarm. Said alarm-output data which said supervisory equipment transmits are received in the interface section of one alarm, and it is characterized by having the means which carries out sequential transmission of this at a daisy chain at the interface section of other alarms. It becomes unnecessary therefore, to be able to make now easily the duplication and modification of each alarm which are prepared in an external alarm, and to prepare the protocol according to individual.

[0011] Moreover, supervisory equipment receives the advice of a failure from the supervisory equipment-ed which constitutes a network (trap). In the network monitoring system with which supervisory equipment transmits alarm-output data to two or more external alarms which consisted of two or more kinds of alarms, and each alarm of each external alarm emits an alarm based on this alarm-output data A means to communalize the alarm-output data to each alarm, and to use a common interface between supervisory equipment and each external alarm, The interface section in which daisy chain connection is possible is used for the interface section by the side of each external alarm. Said alarm-output data which said supervisory equipment transmits are received in the interface section of one external alarm, and it is characterized by having the means which

carries out sequential transmission of this at a daisy chain at the interface section of other external alarms. It becomes unnecessary therefore, to be able to extend an external alarm now easily and to prepare the protocol according to each alarm individual.
[0012] Said supervisory equipment is equipped with a storage means. Moreover, for said storage means The failure generating status file which stores the advice of a failure from said supervisory equipment-ed, The alarm-output configuration file which sets up the alarm level and the address required for a specific alarm when performing [whether an alarm output is performed to said advice of a failure, and] an alarm output, It is characterized by preparing the alarm-output status file which memorizes serially said alarm-output data created from the content of said failure transaction file, and the content of said alarm-output configuration file.
[0013] Moreover, it is characterized by having equipped said supervisory equipment with the interactive I/O device, and having a means to use said interactive I/O device, and to set up and change the content of said alarm-output configuration file. Therefore, the inconvenience by the operator -- become customizable and an alarm continues being emitted at the period under duplication work -- is cancelable.
[0014] Furthermore, it is characterized by having a means to make all the contents memorized to said alarm state file output to said supervisory equipment using said interactive I/O device, and to attain synchronization with said supervisory equipment and said each alarm. Therefore, the inequality produced when the power source of an alarm is turned off is easily cancelable.
[0015]
[Embodiment of the Invention] Hereafter, the operation gestalt of this invention is explained using a drawing. Drawing 1 is the block diagram showing 1 operation gestalt of the equipment configuration of the network monitoring system of this invention, and, as for supervisory equipment-ed and 20, 10 is [supervisory equipment and 30] external alarms in drawing. As for the supervisory equipment 10-ed connected, two or more supervisory equipment-ed of 11-13 exists. Moreover, storage and an interactive I/O device are formed in supervisory equipment 20 (not shown [both]), and as shown in storage at drawing 2 , the failure generating status file 201, the alarm-output configuration file 202, and the alarm-output status file 203 are stored in it. Moreover, the external alarm 30 consists of each alarm of the large-sized network display panel 31, an alarm lamp 32, and buzzer 33 grade, and the interface (IF) section 40 is formed in each alarm at each. Drawing 3 is the block diagram showing the configuration of the IF section 40, it is the gate (gate which consisted of software actually) which 401 carries out transmission and reception of the data of each IF section 40 in the relay circuit for a bypass, and carries out AND of 402 in the unit of transmission and reception, respectively, and it is constituted so that daisy chain connection of two or more IF sections 40 of a base may be made by this.
[0016] If supervisory equipment 20 receives the advice of a failure (trap) from which supervisory equipment 10-ed, to each trap, according to alarm-output setting out beforehand set as the alarm-output configuration file (202 of drawing 2), the alarm levels MJ (measure), MN (minor), and NR (Normal) of advice of a failure and the data of the required address (for example, address of into which part of a panel to perform an alarm display) will be created, and it will transmit to the external alarm 30. Although there is each alarm of an alarm lamp 32 and a buzzer 33 which needs only an alarm level, and the

large-sized network display panel 31 which needs the address and an alarm level among the external alarms 30, with this operation gestalt, data common to each alarm are created and an alarm is generated. Since each IF section 40 is the same configuration, the alarm level and address information which have been sent are transmitted as it is also to an alarm lamp 32 and a buzzer 33, but at an alarm lamp 32 or a buzzer 33, since the address is not needed, it ignores.

[0017] Moreover, as shown in drawing 3 , supervisory equipment 20 and each IF section 40 are connected using the RS-232C connector and the cable, and a relay switch (SW1, SW2) operates so that the relay circuit 401 for a bypass established in the IF section 40 may constitute a bypass circuit, if the power source OFF of the IF section 40 is detected. Moreover, in the IF section 40, AND of transmission and the received data of each IF section 40 is carried out by the gate 402 in the unit of transmission and reception, and as shown in drawing 1 , daisy chain connection of two or more IF sections 40 is made.

[0018] Next, the content of each file stored in the storage of the supervisory equipment 20 of drawing 2 is explained. The failure generating status file 201 stores whether it is under [failure generating] ****** about all the failures notified by the trap. A failure generating condition memorizes the failure notified by the trap. setting out and the comment input of the address information at the time of an operator displaying the alarm-output configuration file 202 on setting out of ON/OFF (/an alarm output is carried out -- don't carry out) of an alarm output performed by using an interactive I/O device, setting out of alarm level (significance) MJ/MN/NR, and a large-sized display panel according to advice of a failure are memorized. These setting out consists of these operation gestalten so that an operator may use the keyboard and mouse of an interactive I/O device and can set up and change as desired. The alarm-output status file 203 is a file which memorizes serially the alarm-output data created and outputted with supervisory equipment 20 from the content of the failure transaction file 201, and the content of the alarm-output configuration file 202, and the alarm-output condition of the alarm level under all address part and current generation is memorized.

[0019] Drawing 4 (A) is a flow chart which shows actuation when supervisory equipment 20 receives the advice of a failure (trap). Supervisory equipment 20 will update the content of the failure generating status file 201 based on the failure notified by the trap, if a trap is received. Next, with reference to the alarm-output configuration file 202, if alarm-output setting out is ON, supervisory equipment 20 will carry out renewal of record of MJ/MN/NR at the alarm-output status file 203 according to an alarm level, and will output an alarm level (MJ/MN/NR) and address information to the external alarm 30. Moreover, when alarm-output setting out of an alarm-output configuration file is OFF, renewal of the alarm-output status file 203 is not performed, and the output to the external alarm 30 is not performed, either.

[0020] Drawing 4 (B) is a flow chart which shows alarm-output setting-out modification processing and alarm synchronization processing. In alarm-output setting-out modification processing, using the keyboard and mouse of an interactive I/O device, the content of the alarm-output configuration file 202 is changed, and an operator updates the content of the alarm-output status file 203 according to alarm-output setting out which read the alarm-output status file 203 to the interactive I/O device, and changed it continuously, and transmits all the information from this file 203 to the external alarm 30.

[0021] Next, alarm synchronization processing is explained. Although it is possible to

perform alarm synchronization processing from both by the side of supervisory equipment 20 and the external alarm 30, in carrying out from a supervisory equipment 20 side, it outputs an alarm synchronization processing demand command to supervisory equipment 20 using the keyboard and mouse of an interactive I/O device. Supervisory equipment 20 transmits all the information that receives this command and is in the alarm-output status file 203 to the external alarm 30, and is made in agreement with the current condition that supervisory equipment 20 has the condition of all the display panels of each alarm, a lamp, and a buzzer. That is, the current condition that supervisory equipment 20 has the alarm which the power source is turned off and became an inequality is synchronized. Moreover, in outputting an alarm synchronization processing demand command from the external alarm 30 side to supervisory equipment 20, an alarm side transmits in the procedure in consideration of the collision of synchronous request of a slave side in the slave side of the multidrop of RS232C (using polling, a selecting method, etc.).

[0022] Next, the example of this invention is explained using drawing 1 and drawing 2 . For example, when "xxx basic trunk failure" occurs with supervisory equipment 10-ed, "xxx basic trunk failure generating" is notified to supervisory equipment 20 by the advice of a failure (trap). Supervisory equipment 20 stores "xxx basic trunk failure generating" in the failure generating status file 201, if this trap is received. And the record which corresponds to xxx basic trunk failure with reference to the alarm-output configuration file 201 is searched.

[0023] Supervisory equipment 20 records the print-out outputted to the external alarm 30 on the alarm state output file 203 according to setting out of the output of this record, an alarm level, and the address, when the record applicable to xxx basic trunk failure is found. In the alarm-output configuration file 202, ON and an alarm level create the data (MJ generating / MN-less/NR nothing and xxx) as information which outputs them to the external alarm 30 since MJ (major) and the address are xxx(es), and an output transmits. Each external alarm 30 receives this data (MJ generating, xxx), with each alarm, is displayed and carries out singing of the alarm.

[0024] Next, actuation of each alarm is explained. With the large-sized network display panel 31, data (MJ generating, xxx) are received and the circuit which the address of xxx shows is displayed in MJ color (for example, red). Moreover, in an alarm lamp 32, data (MJ generating, xxx) are received and the lamp of MJ color (for example, red) is turned on (since it is unnecessary, the address is disregarded). Furthermore, at a buzzer 33, data (MJ generating, xxx) are received and singing of the buzzer is carried out by MJ (it is continuation singing until it restores for example, with the maximum sound volume) (since it is unnecessary, the address is disregarded). Moreover, in another room (another floor, separate building), since the same external alarm 30 by which daisy chain connection was made between the IF sections 40 is installed and each alarm can receive data (MJ generating, xxx), alarm actuation is performed similarly.

[0025] Next, for the reason under work etc., the actuation which stops / resumes xxx basic trunk failure alarm under present output is explained using drawing 2 and drawing 4 . An operator changes the output of xxx basic trunk failure of the alarm-output configuration file 202 of drawing 2 from ON at OFF using the keyboard and mouse of an interactive I/O device. Moreover, the comment of a purport under construction now is added if needed. According to setting-out modification of this output, from the failure

generating status file 201 and the alarm-output configuration file 202, supervisory equipment 20 updates the alarm state of the address of the alarm-output status file 203 concerned, and transmits it to the external alarm 30.

[0026] Since it not only restores the alarm state of the address of the alarm-output status file 203 concerned, but it may be carrying out the alarm display including two or more fault conditions in the one address like for example, bbb equipment failure (address bbb) and ccc card failure (address bbb) when restoring alarm cutoff, it is necessary to judge again the fault condition generated by then in the corresponding address, and to update the alarm-output status file 203. By the same approach, only not only in setting-out modification of the output of an alarm, and the alarm level of the alarm-output configuration file 202 and setting-out modification of the address can be made (in the flow of alarm-output modification, it is supposed that simple is carried out and all alarms are outputted again).

[0027] Finally synchronization processing is explained. In this case, all the contents of the alarm-output status file 203 are outputted, and it is made in agreement with the current condition that supervisory equipment has the condition of all the display panels of each alarm, a lamp, and a buzzer. Although it is not easy to become supervisory equipment 20 and an inequality and to synchronize this when the power source is turned off for any of an alarm they are at the time of the data transmission since it does not have a means to memorize the transmitted alarm-output data in the conventional network monitoring system , with this operation gestalt , it can synchronize easily by make all the contents of the alarm-output status file 203 output .

[0028] Drawing 5 is the block diagram showing other operation gestalten of this invention. Although considered as the configuration which forms the IF section 40 in the external alarm 30 with the operation gestalt shown in drawing 1 , it is good also as a configuration which makes the IF section 40 become independent of the external alarm 30 as shown in drawing 5 .

[0029] Moreover, drawing 6 is the block diagram showing the operation gestalt of further others of this invention. Although considered as the configuration which forms the IF section 40 in each alarm, respectively with the operation gestalt shown in drawing 1 , it is good for the external storage 30 installed one set at a time in each ** (a floor, ridge) by making the number of connectors of the IF section 40 into plurality also as a configuration which forms one IF section 40, respectively.

[0030]

[Effect of the Invention] As explained above, the network monitoring system of this invention communalizes the alarm-output data to each alarm, and since [the interface section] daisy chain connection is made, a system, such as duplication and modification of an alarm, and forming an external alarm in two or more rooms further, can be extended easily. It becomes unnecessary moreover, for supervisory equipment to create alarm-output data for every alarm. Moreover, an operator can control an alarm by changing the content of the alarm-output configuration file according to an individual, and inconvenience, such as being continued by emitting this an alarm etc. during a duplication period, can be canceled. Since it has a means to make all the contents furthermore memorized to the alarm state file output, and to attain synchronization with supervisory equipment and each alarm, there is effectiveness that the inequality of the alarm state of an alarm is easily cancelable etc.

CLAIMS

[Claim(s)]

[Claim 1] Supervisory equipment receives the advice of a failure from the supervisory equipment-ed which constitutes a network (trap). In the network monitoring system with which supervisory equipment transmits alarm-output data to the external alarm which consisted of two or more kinds of alarms, and each alarm emits an alarm based on this alarm-output data A means to communalize the alarm-output data to each alarm, and to use a common interface between supervisory equipment and each alarm, The interface section in which daisy chain connection is possible is used for the interface section by the side of each alarm. Network monitoring system characterized by having received said alarm-output data which said supervisory equipment transmits in the interface section of one alarm, and having the means which carries out sequential transmission of this at a daisy chain at the interface section of other alarms.

[Claim 2] Supervisory equipment receives the advice of a failure from the supervisory equipment-ed which constitutes a network (trap). In the network monitoring system with which supervisory equipment transmits alarm-output data to two or more external alarms which consisted of two or more kinds of alarms, and each alarm of each external alarm emits an alarm based on this alarm-output data A means to communalize the alarm-output data to each alarm, and to use a common interface between supervisory equipment and each external alarm, The interface section in which daisy chain connection is possible is used for the interface section by the side of each external alarm. Network monitoring system characterized by having received said alarm-output data which said supervisory equipment transmits in the interface section of one external alarm, and having the means which carries out sequential transmission of this at a daisy chain at the interface section of other external alarms.

[Claim 3] Network monitoring system according to claim 1 or 2 characterized by providing the following The failure generating status file which equips said supervisory equipment with a storage means, and stores the advice of a failure from said supervisory equipment-ed in said storage means The alarm-output configuration file which sets up the alarm level and the address required for a specific alarm when performing [whether an alarm output is performed to said advice of a failure, and] an alarm output The alarm-output status file which memorizes serially said alarm-output data created from the content of said failure transaction file, and the content of said alarm-output configuration file

[Claim 4] Network monitoring system according to claim 3 characterized by having equipped said supervisory equipment with the interactive I/O device, and having a means to use said interactive I/O device, and to set up and change the content of said alarm-output configuration file.

[Claim 5] Network monitoring system according to claim 3 characterized by having a means to make all the contents memorized to said alarm state file output to said supervisory equipment using said interactive I/O device, and to attain synchronization

with said supervisory equipment and said each alarm.

[Translation done.]

METHOD AND SYSTEM FOR TESTING THE VALIDITY OF SHARED DATA IN A MULTIPROCESSING SYSTEM

BACKGROUND OF THE INVENTION

[0001] The present disclosure relates generally to a method and apparatus for testing the validity of shared data in a multiprocessing system, and particularly to a method and apparatus for random testing data block concurrency, execution sequence and serialization in multiprocessing systems in response to pseudo-random instruction streams.

[0002] In a typical multiprocessing environment, a number of central processing units (CPUs) share data residing in a particular location of the main/cache memory. These CPUs may be required to obey certain rules such as block concurrence, execution order and serialization. The block concurrence rule states that while a given CPU is fetching from or storing into a block of shared memory, the operation must be done in indivisible units. For example, if a fetch unit is four bytes, then the CPU is not allowed to fetch the data in two or one byte increments. The execution order rule requires that all stores and fetches are executed in order. In addition, the serialization rule refers to the ability of the CPU to use pre-fetched data until the CPU serializes, at which point it discards all pre-fetched data and completes all pending stores before it executes new fetches.

[0003] An access by an instruction to a memory location may be a fetch, a store or an update. In the case of the fetch access, the instruction does not alter the contents of the memory location. In contrast, the store access type instruction modifies the contents of the

memory location. An instruction that fetches from and then stores into the same memory location performs an update access type. An update access operation can be viewed as a fetch operation followed by a store operation.

[0004] Several random test methodologies regarding cache coherence of the weakly-ordered (e.g., Reduced Instruction Set Computer (RISC)) architecture, such as PowerPC, have been reported in the literature. In the data-coloring technique, each store operation writes unique data into the common memory location. In this method if instructions storing into the shared locations are allowed to access a storage location consisting of a combination of shared and non-shared locations, one has to account for the worst case and thus limit store operations affecting these shared locations to 256 (maximum number of distinct data colors that can be represented with one byte). Therefore, the number of instructions of a given CPU that could write into the shared location becomes severely limited as the number of CPUs increases. The use of monotonic input data for multiprocessors and analyzing the terminal results is a special case of the data coloring technique. The false sharing method partitions the shared memory location such that different CPUs store into and fetch from non-overlapping locations. Other approaches force synchronization instructions before each store operation that stores into the shared location. However, these techniques concentrate on weakly-ordered architectures such as PowerPC and do not consider when arithmetic and logic operations engage in register-to-memory, memory-to-register, register-to-memory or memory-to-memory data transfer operations. Furthermore, the fact that an instruction stream and translation tables can be subject to modifications due to the actions of some CPUs are not taken into account.

[0005] In the case of a firmly-ordered architecture (e.g., Complex Instruction Set Computer (CISC)), such as IBM's System 390 architecture, a store operation executed by an instruction can take place (as seen by other CPUs) only after all the preceding store operations

are completed. Furthermore, both logical and arithmetic operations of the IBM System 390 architecture can involve register-to-memory, memory-to-register and memory-to-memory operations. However, unless another CPU observes, one cannot conclude that any of the store operations of a given CPU are completed. Hence, determining the validity of the fetched data from a memory location is complicated when more than one CPU may have stored data into that location. Data fetched by a CPU_i from a memory location M which has been subject to a number of store operations by different CPUs may not adhere to certain rules such as the block concurrence, execution order and serialization rules.

SUMMARY OF THE INVENTION

[0006] In one embodiment, a method for testing the validity of shared data in a multiprocessing system is disclosed. The method comprises receiving at a first central processing unit a list of fetch and store instructions associated with blocks in a shared memory location. The list includes a data value, a central processing unit identifier and a relative order associated with the instructions. In addition, one of the data values associated with one of the instructions was stored by a memory-to-memory, memory-to-register or register-to-memory operation. Further, one of the central processing unit identifiers associated with one of the instructions is an identifier corresponding to one of a plurality of central processing units that have access to the shared memory location including the first central processing unit. A fetch operation is performed at a block in the shared memory location from the first central processing unit. Fetched data is received at the first central processing unit in response to the performing, where the fetched data was stored by one of the plurality of central processing units. The method verifies that the fetched data conforms to a block concurrency rule in

response to the list and that the fetched data conforms to a serialization rule in response to the list.

[0007] In another embodiment, a computer program product for testing the validity of shared data in a multiprocessing system is disclosed. The computer program product comprises a storage medium readable by a processing circuit and storing instructions for execution by the processing circuit for performing a method. The method comprises receiving at a first central processing unit a list of fetch and store instructions associated with blocks in a shared memory location. The list includes a data value, a central processing unit identifier and a relative order associated with the instructions. In addition, one of the data values associated with one of the instructions was stored by a memory-to-memory, memory-to-register or register-to-memory operation. Further, one of the central processing unit identifiers associated with one of the instructions is an identifier corresponding to one of a plurality of central processing units that have access to the shared memory location including the first central processing unit. A fetch operation is performed at a block in the shared memory location from the first central processing unit. Fetched data is received at the first central processing unit in response to the performing, where the fetched data was stored by one of the plurality of central processing units. The method verifies that the fetched data conforms to a block concurrency rule in response to the list and that the fetched data conforms to a serialization rule in response to the list.

[0008] In a further embodiment, a system for testing the validity of shared data in a multiprocessing system is disclosed. The system comprises a plurality of central processing units including a first central processing unit and a shared memory location accessed by the plurality of central processing units. The system further comprises application software located on the first central processing unit for implementing a method comprising receiving at the first central processing unit a list of fetch and store instructions associated with blocks in the shared memory

location. The list includes a data value, a central processing unit identifier and a relative order associated with the instructions. In addition, one of the data values associated with one of the instructions was stored by a memory-to-memory, memory-to-register or register-to-memory operation. Further, one of the central processing unit identifiers associated with one of the instructions is an identifier corresponding to one of the plurality of central processing units that have access to the shared memory location including the first central processing unit. A fetch operation is performed at a block in the shared memory location from the first central processing unit. Fetched data is received at the first central processing unit in response to the performing, where the fetched data was stored by one of the plurality of central processing units. The method verifies that the fetched data conforms to a block concurrency rule in response to the list and that the fetched data conforms to a serialization rule in response to the list.

BRIEF DESCRIPTION OF THE DRAWINGS

[0009] Referring to the exemplary drawings wherein like elements are numbered alike in the accompanying Figures:

[0010] FIG. 1 is a block diagram of a high level process flow for an exemplary embodiment of the present invention;

[0011] FIG. 2 is an exemplary embodiment of an algorithm that may be utilized to test for block concurrence;

[0012] FIG. 3 is an exemplary embodiment of an algorithm for testing machine compliance to serialization rules; and

[0013] FIG. 4 is a block diagram of an exemplary system for testing the validity of shared data in a multiprocessing system.

DETAILED DESCRIPTION OF THE INVENTION

[0014] An exemplary embodiment of the present invention provides a method of testing the validity of data fetched by a given CPU from storage locations that are subject to store and/or update operations by multiple CPUs. This includes both load and store operations as well as arithmetic and logical operations. The conformance of a machine to the block concurrence, order of execution and serialization rules is verified. In an exemplary embodiment of the present invention, it is assumed that prior to the analysis, the order in which the CPUs access the shared location is not known (i.e., time relationship among the CPUs is not available). FIG. 1 is a block diagram of a high level process flow for an exemplary embodiment of the present invention. The high level process flow depicts an exemplary method for testing of the validity of shared data in a multiprocessing system. At step 102, the shared memory locations are identified. At step 104, a pseudo-random test instruction stream is built for each CPU. These pseudo-random streams include instructions that fetch and store into the shared memory locations. A table of shared information (TOSHI) is built at step 104. When an instruction is selected to access a shared location, a TOSHI entry is created for that instruction. A TOSHI entry includes information such as the instruction address, the target address of the data, the source address of the data, any registers involved in the shared location and any associated flags (e.g., an execution flag). It is assumed that instructions are executed by their respective CPUs and not in a simulation environment. If simulation is available the TOSHI will contain less information.

[0015] In an embodiment of the present invention, a single operation accessing a shared location may simultaneously write to or read from a combination of shared and non-shared locations. For example, if the instruction stores four bytes into the memory, the following combinations are possible: all four bytes are part of a shared location; three bytes are part of a shared location and the other byte is part of a non-shared location; two bytes are part of a shared location and the other two bytes are part of a non-shared location; one byte is part of a shared location and the other three bytes are part of a non-shared location; or all four bytes are from a non-shared location. In other words, an embodiment of the present invention allows processors to share up to the bit level. In addition, in an embodiment of the present invention, when an instruction fetches data from a shared location, the data is protected (i.e., is not overwritten) until it is made observable by another store operation. For example, if instruction "I" fetches from a shared location "B" and stores the data into "A", where "A" is a register, the register is marked (via TOSHI flags) that it contains data that needs to be observed. Therefore, until another store-type instruction uses the register and stores into a non-shared location, no instruction can overwrite the data contained by this register. To maintain the randomness of the instruction stream, a number of randomly selected instructions is inserted into the stream between the point where an instruction fetches from a shared location and when the fetched data is made observable.

[0016] At step 106, the stream of instructions are passed to their respective CPUs. The streams are then executed by their respective CPUs. New pseudo random streams of instructions are built and passed to the CPUs if no error is detected in the previous streams. At step 108, relevant data information is gathered from the execution as directed by the entries in the TOSHI. In an exemplary embodiment of the present invention, the relevant data structure extracted from TOSHI includes lists containing the data stored by each CPU to each shared

memory location and the addresses of instructions that are storing into or fetching from the shared location, the addresses of the source or observable (i.e., known or resultant) data, and those instructions that cause serialization. At step 110, conformance to block concurrency rules is verified. The block concurrency rule checks that a CPU stores and fetches in indivisible units (e.g., half-word, double-word). In addition, at step 110, conformance to the execution order rules is verified. Execution rules refer to the requirement that all stores and fetches to the shared memory location should be performed in order. Conformance to serialization rules is verified at step 112. Serialization occurs when a CPU is required to discard all pre-fetched data and completes pending data stores. An embodiment of the present invention verifies that a CPU has performed the serialization and has discarded all pre-fetched data. At step 114, the process, starting at step 102, is repeated if there was no error in steps 110 and 112. If there was an error in step 110 or step 112, the process depicted in FIG. 1 is exited with an error message.

Fetch/Store Operations and Block Concurrency Verification

[0017] In this section, an embodiment of the present invention that includes the ability to verify the block concurrence of memory locations, altered by register-to-memory and memory-to-memory store operations (e.g., IBM S/390 *rx* and *ss-format*, respectively) of different CPUs is described. Let M be a memory location which may be altered by a number of CPUs. The goal is to test if data fetched by CPU_i from M is valid. It is assumed that immediately after a CPU stores into M all other CPUs can reference M and use the new value. A limitation on the observability of M is due to the fact that the contents of M can be observed only when a given CPU accesses M by executing a fetch-type instruction. For each CPU_i that stores into M, a set or list $L_{CPUi}(M)$, containing the stored data by the CPU is created. Once a particular CPU_j fetches data D from M, an effort is made to determine:

1. which CPU has stored the fetched data; and

2. if the search in step 1 above is successful, then the uniqueness of the data (i.e., whether it was stored by a single CPU_k) is checked. If the data is found to be unique, an attempt is made to reduce the set L_{CPUk}.

In other words, the existence of D in all $L_{CPU}(M)$s is investigated. An invalid data D has been fetched if D is not found in these sets. On the other hand, if D is a member of one or more of the $L_{CPU}(M)$s, determining the CPU that stored D is performed next. In an exemplary embodiment of the present invention, the way that such a CPU can be identified is if D exists only in a single $L_{CPU}(M)$. If it is determined that CPU_i stored D, then L_{CPUi} is reduced, if possible, by removing all entries that have been stored by CPUi prior to D from the set. Such reduction can be performed since all preceding store operations of CPU_i must have been performed prior to storing D (conceptual sequence). The resulting list, L_{CPUi}, is then accessed to evaluate the data from the next fetch instruction that includes data stored by CPU_i.

[0018] An exemplary embodiment of an algorithm that may be utilized to test block concurrence is shown in FIG. 2. Note that the size of M depends upon the instruction being executed and is an integral of the specified block concurrence. At step 202 in the algorithm, a loop is initiated to retrieve instructions, denoted as "I" in FIG. 2, from the stream, denoted as "S" in FIG. 2. The stream includes a sequence of any number of instructions and may include instructions that access the shared locations. During stream execution, the following information is collected: instructions fetching from or storing into shared locations, the data value being stored, the data value being stored/fetched (if known), a CPU identifier corresponding to the CPU performing the instruction and the address of the instruction relative to other instructions in the stream. For each fetch instruction located, as tested at step 204, the "FOUND" flag is set to

false, the "ValidData" flag is set to false and the "UniqueData" flag is set to true. Each fetch instruction also includes a data value, "*D*", that was fetched. A loop, starting at step 206, is performed for each CPU that utilizes the shared data location, starting with the first CPU, while the "UniqueData" flag is still true. If the "UniqueData" flag is set to false (i.e., the data value "*D*" has been stored by more than one CPU), then the algorithm depicted in FIG. 2 continues to analyze the next instruction. If the data, *"D"*, is located in the list of stores associated with the current CPU being searched, L_{CPUi}, then at step 208, the "ValidData" flag is set to true. This means that the data being verified, "*D*" has been located and identified as being stored on one of the CPUs, and that "*D*" is valid data. Next, another loop is performed at step 210 to determine if the data, "*D*", is also located in a list associated with another CPU. If this is the case, then the "UniqueData" flag is set to false. Otherwise, at step 212, the list associated with the CPU where the data, *"D"* was located is reduced based on the position of "*D*" in the list. All data store entries prior to "*D*" in the list L_{CPUi} are removed. If the data is not located in one of the CPU lists, then the "ERROR" flag is set to true, indicating that the data is invalid and that the block concurrence rules were violated.

[0019] For example, assume that CPU_1 and CPU_2 store into shared data storage, *M*, the following data values, or "*D's*".

L_{CPU1}	L_{CPU2}
a	a
b	a
c	a
c	b
d	g
e	

f

For purposes of this example, also assume that CPU_3 repeatedly executes fetch operations and retrieves "a", "c", "b", "g", and "f" from the shared location. When CPU_3 fetches "a", the data fetched may have came from either CPU_1 or CPU_2. Therefore, although L_{CPU1} and L_{CPU2} cannot be reduced, the data is determined to be valid. When CPU_3 fetches "c", the fact that CPU_1 is the only one that stored "c" into M implies that "c" is valid data. In addition, the data values "a" and "b", stored by CPU_1, are overwritten and are no longer valid data values. Therefore, the lists are reduced as follows.

L_{CPU1}	L_{CPU2}
c	a
c	a
d	a
e	b
f	g

[0020] Similarly, when CPU_3 reads "b", the sequence of "a's" stored by CPU_2 is overwritten and cannot be read. The resulting lists of data are as follows.

L_{CPU1}	L_{CPU2}
c	b
c	g
d	

[redacted]

e

f

[0021] When CPU_3 retricves "g", the following lists of data result after "b", which has been overwritten by CPU_2 is deleted from the list.

L_{CPU1}	L_{CPU2}
c	g
c	
d	
e	
f	

[0022] When CPU_3 reads "f", then "c", "d" and "e" are no longer valid data values to be fetched because they have been overwritten by CPU_1, and the lists of expected data are reduced to the following.

L_{CPU1}	L_{CPU2}
f	g

[0023] In another example, utilizing an exemplary embodiment of the present invention, it is assumed that the store operations of CPU_1 and CPU_2 happen in a particular order. In this example, CPU_1 and CPU_2 execute the following store operations.

CPU_1	CPU_2
	$M \leftarrow d$
$M \leftarrow b$	
$M \leftarrow a$	
	$M \leftarrow c$
$M \leftarrow d$	

[0024] Further, in this example, CPU_3 accesses M by executing a sequence of fetch operations. The lists of data observable by for CPU_3 are as follows.

L_{CPU1}	L_{CPU2}
b	d
a	c
d	

[0025] First, suppose that CPU_3 fetches "d". Since "d" is stored by both CPU_1 and CPU_2 (i.e., it is not uniqc data), the list of observable data does not get reduced. Next, if CPU_3 reads "c" it can be concluded that the fetched data is both valid and unique because "c" appears in only one list. The "d" stored by CPU_2 will never be fetched again by CPU_3 as stated by the conceptual sequence rules. Therefore, the lists of data that may be observed by CPU_3 are reduced as depicted below.

L_{CPU1}	L_{CPU2}

b c

a

d

[0026] Next, in this example, CPU_3 retrieves "d". At this time "d" is both valid and unique data. Fetching "d" implies that CPU_3 will not be able to observe "c" since all fetches must in order. Also, CPU_3 will not be able to fetch the "b" and "a" stored by CPU_1 again and the lists of observable data for CPU_3 are reduced to the following.

L_{CPU1}	L_{CPU2}
d	

[0027] Next, is an example where the block concurrency is violated. Suppose the list of CPU_1 and CPU_2 stored data is the following.

L_{CPU1}	L_{CPU2}
abcd	ijkl
efgh	

A block concurrency/execution order violation would occur, for example, if CPU_3 fetches "ijgh".

Arithmetic/Logical Operations and Block Concurrence

[0028] An exemplary embodiment of the present invention checks the conformance of logical and arithmetic operations, that fetch data from shared memory locations, to the block

concurrence rules. Such operations are in the form of "A ← A <op> B", where "<op>" is a logical or arithmetic operator, "A" is retrieved from and then stored into a register, and "B" is fetched from a shared memory location considered. When B is from a shared memory location, the method creates observable points such that the value loaded into A is available for future analysis. Since both the old and final values of A are known, an exemplary embodiment of the present invention attempts to determine some or all of the bits of B. The method uses reverse operations to find some or all of the bits fetched from the shared locations. Although a strong conclusion cannot be made just by knowing only a portion of B's bits, such portion of the data may be utilized to deduce whether a given CPU "possibly" stored B. The search is performed as described in the previous section except that certain bits of B may not be known. As a result, decisions are made based on possibilities.

[0029] In the following example for determining all or a portion of the bits of B, B is fetched from a shared memory location and A is retrieved from and then stored into a register. The bits of A are referred to as "a_i" and the bits of B are referred to as"b_i". Also, in the example, "ai_{old}" is the value of a_i before the store operation is executed and "ai_{new}" is the value of a_i after the store is completed. Exemplary calculations for determining all or a portion of the bits in B are described below for logical operations and arithmetic operations.

Logical Operations

XOR

$$ai_{new} \Leftarrow ai_{old} \text{ XOR } b_i$$

$$b_i = ai_{new} \text{ XOR } ai_{old}$$

AND

$$ai_{new} \Leftarrow ai_{old} \text{ AND } b_i$$

case 1:

$(ai_{new} = 1)$ AND $(ai_{old} = 1) \Longrightarrow b_i = 1$

case 2:

$(ai_{new} = 0)$ AND $(ai_{old} = 1) \Longrightarrow b_i = 0$

case 3:

$(ai_{new} = 0)$ AND $(ai_{old} = 0) \Longrightarrow b_i = u$

case 4:

$(ai_{new} = 1)$ AND $(ai_{old} = 0) \Longrightarrow b_i = u$;

where "u" denotes unknown.

OR

$ai_{new} \Leftarrow ai_{old}$ OR b_i

case 1:

$(ai_{new} = 0)$ AND $(ai_{old} = 0) \Longrightarrow b_i = 0$

case 2:

$(ai_{new} = 1)$ AND $(ai_{old} = 0) \Longrightarrow b_i = 1$

case 3:

$(ai_{new} = 1)$ AND $(ai_{old} = 1) \Longrightarrow b_i = u$

case 4:

$(ai_{new} = 0)$ AND $(ai_{old} = 1) \Longrightarrow b_i = u$

Arithmetic Operations

Logical Addition

$A_{new} \Leftarrow A_{old} + B$

$B = A_{new} - A_{old}$

where A_{new} is the value of A after the store is completed and A_{old} is the value of A before the store is executed

Logical Subtraction:

$A_{new} \Leftarrow A_{old} - B$

$B = A_{old} + A_{new}$

Multiplication:

The product of multiplying A and B, where both are n bit operands, is placed in a field, "P", of $2*n$ bits. The most significant bit of the product represents the sign. P may be made up of an even /odd pair of registers (e.g., in the IBM S/390 architecture).

P', A' and B' represent the magnitude of P, A, and B, respectively. Also, $s(X)$ is the sign of X (note that an attempt to find B should not be made if A is equal to zero) and "u" stands for unknown.

$P \Leftarrow A*B$

$B' = P'/A'$

case 1: $(s(P) = 1)$ AND $(s(A = 1) \Rightarrow s(B) = 1$

case 2: $(s(P) = 0)$ AND $(s(A = 1) \Rightarrow s(B) = 0$

case 3: (s(P) = 0) AND (s(A = 0) ⇒ s(B) = u

case 4: (s(P) = 1) AND (s(A = 0) ⇒ s(B) = u

Division:

$Q' \leftarrow A_{old}/B$, where Q' is the quotient

$B' = A'/Q'$

case 1: (s(Q) = 1) AND (s(A = 1) ⇒ s(B) = 1

case 2: (s(Q) = 0) AND (s(A = 1) ⇒ s(B) = 0

case 3: (s(Q) = 0) AND (s(A = 0) ⇒ s(B) = u

case 4: (s(Q) = 1) AND (s(A = 0) ⇒ s(B) = u

[0030] Additionally, to facilitate the observability of data in a shared memory location, M, after an update operation, A ← A <op> B, stores into M, B may be chosen such that portions of the data retrieved by subsequent fetch instructions are "predictable". Logical operations may be utilized to store predictable data into M as shown below.

A ← A AND B : $(b_i = 0) \Rightarrow (a_i = 0)$

A ← A OR B : $(b_i = 1) \Rightarrow (a_i = 1)$

[0031] In an exemplary embodiment of the present invention, the above list of logical and arithmetic operations may be expanded to include other operations. For example, an embodiment may be extended to nonlogical arithmetic and take into account underflow and overflow by masking the status information bits.

[0032] When a shared location is both a source and a target, the analysis becomes more difficult. However, the stream may be biased so that meaningful information may be determined from subsequent fetches. For example, when A (both a target and a source) is from

a shared location and B (a source) is from a non-shared location, the logical operations of AND and OR and use B as a mask. If the data stored by this update instruction is observed by any CPU, the mask is used to assure if certain bits are on (for the OR operation) or off (for the AND operation).

[0033] An embodiment of the present invention also determines if instruction fetching, from shared locations, obeys specific block concurrence rules as described by the architecture. For example, in the IBM/390 architecture, instruction fetching must be half word block concurrent. If portions of the instruction stream belong to the shared memory, it is possible that some of the CPUs will alter the instruction stream by executing store or update type instructions. Furthermore, the fact that one cannot observe the fetched instructions adds to the complexity of analyzing fetched instructions. In some system architectures (e.g., the IBM S/390), the length of an instruction can be two, four, or six bytes. Once a CPU stores into the stream, one or more instructions may get modified. As an example, storing a six-byte data into the stream can replace three two-byte, one two-byte and one four-byte, or one six-byte instruction(s). As a result of storing into the stream area, some of the executed instructions may not appear in the original stream of instructions since certain CPUs may have altered the stream by storing data into the shared locations. The combination of possible instructions and the operands used by these instructions may become extremely large. However, such combinations can be dramatically reduced by associating certain instructions with particular operands. Without losing generality, further reduction can also be realized by selecting, randomly, the unknown instructions in a given stream to a specific group of instructions. As an example, arithmetic instructions may be selected in one stream and logical instructions in another. To analyze the result (if any) of the fetched instruction, an embodiment of the present invention attempts to determine, or guess, the instruction that would produce the result and then, if such instruction

does not belong to the original instruction stream, whether the instruction was inserted into the stream by one of the CPUs. The problem then becomes similar to those described previously except that at this time the fetched data, which is an instruction, was guessed rather than observed.

[0034] An example of how this is may be accomplished follows. Let S_{cpui} and $A(S_{cpui})$ be the stream of instructions and their addresses, respectively, that CPU_i is executing as described in the table below.

$A(S_{cpu1})$	S_{cpu1}	$A(S_{cpu2})$	S_{cpu2}
0000	$R_2 \leftarrow R_2 + R_3$	0010	$0002 \leftarrow 1F21$
0002	$R_2 \leftarrow R_2 + R_1$		
0004	$R_2 \leftarrow R_2 + R_2$		

For simplicity, all instructions affecting R_2 are restricted to performing only arithmetic operations. Let the initial values of R_1, R_2, and R_3 be 1, 2, and 3, respectively. In addition, it is assumed that R_2 was observed to have the values of 5, 4, and 8. Along with the original stream generated for CPU_1, the list of data stored into the stream by CPU_2 is available. In this example, such a list includes 1*F*21 which is equivalent to $R_2 \leftarrow R_2 - R_1$. Based on the instruction stream generated for CPU_1, the instruction inserted into that stream by CPU_2, and the different values contained by R_2, the algorithm checks to see if the executed instructions were fetched properly. The initial value contained by R_2 was 2. The second value, 5, can be guessed to be obtained as a result of the add instruction at 0002. It can be assumed that $R_2 \leftarrow R_2 + R_1$ was not executed since such an operation would produce a result of 6. Therefore, guessing that a subtract operation may have been performed by CPU_1, the algorithm checks the list of data inserted into the stream by CPU_2. Such data confirms the guessing that a subtract operation should be performed if the

inserted instruction is exccuted. A comparison of the result that could be produced by the inserted instruction and the value contained in R_2 assure that such an instruction was fetched and executed properly. The above analysis may be extended to the four and six-byte long instructions. It is important to emphasize, that instructions whose lengths are more than two-bytes can result due to different store/update operations of different CPUs. Therefore, the analysis should include invcstigating if such instructions were due to the actions of a single or different CPUs.

[0035] Additionally, locating operands and/or instructions, addressed by the instructions of a stream, may require the use of a translation process before storing into or fetching from implicitly specified locations. The translation process of the IBM S/390 system architecture, for example, involves using tables such as the segment-table and page-table. Similar to the instruction stream case, a translation table designated for a given CPU_i may belong to the shared location. Hence, under certain conditions, one or more CPUs may manage to modify entries of CPU_i's translation table. As a result, some data will be fetched from or stored into different addresses from thosc specified by the stream that CPU_i initially started executing. Testing th validity of the data fetched from translation tables when CPUs have the capability of altering certain translation table entries can basically be performed as described previously. The process of testing the validity of such data includes invcstigating whether the modified entries of the translation table are fetched according to the block concurrence rulcs.

Execution Sequence and Multiple Shared Locations

[0036] In an exemplary embodiment of the present invention, verification of the machine compliance to thc execution order is based on the fact that all stores (and all fetches) must be executed in order. Once CPU_i observes data stored by CPU_j, an exemplary embodiment of the

present invention deletes from the observable lists of CPU_i any data stored by CPU_j prior to this store operation. For each CPU that fetches data from shared-locations, copies of all lists of data stored into the shared-locations and the sequence of store operations of each CPU are created. Once CPU_i fetches data from a shared-location and CPU_j is identified to be the processor that stored the data, the lists of stored data by CPU_j (which can be observed by CPU_i) may be reduced. The fact that the sequence of all store operations of CPU_j is known allows a reduction in more than one list (i.e., at different memory locations) of data stored by CPU_j.

[0037] For example, let M_1, M_2, and M_3 be three different shared-memory locations in which CPU_1, CPU_2, and CPU_3 stored the following data.

M_1			M_2			M_3		
CPU_1	CPU_2	CPU_3	CPU_1	CPU_2	CPU_3	CPU_1	CPU_2	CPU_3
a	b	c	a	x	r	c	y	x
g	e	f	b	e	f	d	x	y
d	i	j	s	n	j	n	p	s

In addition, the example includes a sequence of store operations of CPU_2 as follows:

$M_1 \Leftarrow b$

$M_2 \Leftarrow x$

$M_3 \Leftarrow y$

$M_3 \Leftarrow x$

$M_2 \Leftarrow e$

$M_2 \Leftarrow n$

$M_1 \Leftarrow e$

$M_3 \Leftarrow p$

$M_1 \Leftarrow i$

[0038] If CPU_i observes "p" from M_3, more than one list of data stored by CPU_2 may be reduced after verifying that "p" was stored by CPU_2 alone (i.e., is unique and valid data). An exemplary embodiment of the present invention reduces the lists of data stores in the following manner by taking into account that observing the "p" implies that all stores of CPU_2 prior to "p" must have happened. After "p" is observed, CPU_i will not observe "y" and "x" from M_3. It is also known that $M_1 \Leftarrow b$ and $M_1 \Leftarrow c$ (by CPU_2) took place before $M_3 \Leftarrow p$. Therefore, the value "b" (written by CPU_2) is overwritten by $M_1 \Leftarrow e$. In addition, after "p" is observed, CPU_i will never observe "x" and "e" from M_2 because the CPU_2 operation of $M_2 \leftarrow n$ happened before the $M_3 \Leftarrow p$ operation. An exemplary embodiment of the present invention reduces the lists of data stored into M_1, M_2 and M_3 after "p" is observed as follows.

	M_1			M_2			M_3		
CPU_1	CPU_2	CPU_3	CPU_1	CPU_2	CPU_3	CPU_1	CPU_2	CPU_3
a	e	c	a	n	r	c	p	x
g	i	f	b		f	d		y
d		j	s		j	n		s

[0039] In addition, in an exemplary embodiment of the present invention, verification of the machine-compliance to the left to right order is demonstrated by using the move character (MVC) operation, as an example operation. The MVC operation moves a number of consecutive blocks from one memory location to another. For example, let each of M_1, M_2, and M_3 be one block. If CPU_2 executes a MVC that stores into these blocks, and M_3 is determined to contain data from that MVC operation, M_1 and M_2 should not contain any data of previous stores from CPU_2. M_1 and M_2 should have the data stored by the MVC operation or any other store operation that may take place after the MVC. For example, M_1, M_2 and M_3 are three different shared-memory locations into which CPU_1, CPU_2 and CPU_3 store the following data.

	M_1			M_2			M_3		
CPU_1	CPU_2	CPU_3	CPU_1	CPU_2	CPU_3	CPU_1	CPU_2	CPU_3
a	b	c	a	x	r	c	y	z
	e	f	b	e	f		p	
								y
	i		s	n	j		x	s

[0040] In addition, the example includes a single instruction performed by CPU_2, MVC: $M_1 M_2 M_3 \Leftarrow$ "eex", that causes three different stores:

$M_1 \Leftarrow$ e

$M_2 \Leftarrow$ e

$M_3 \Leftarrow$ x.

[0041] Finally, the example assumes that the sequence of store operations of CPU_1 is as follows.

$M_1 \Leftarrow$ b

$M_2 \Leftarrow$ x

$M_3 \Leftarrow$ y

$M_3 \Leftarrow$ p

$M_2 \Leftarrow$ e

$M_2 \Leftarrow$ n

$M_1 \Leftarrow$ e

$M_3 \Leftarrow$ x

$M_1 \Leftarrow$ i

[0042] If CPU_i fetches "x" from M_3, then an exemplary embodiment of the present invention reduces the lists of CPU_2 observable data in the following manner. The list of CPU_2 observable data from M_3 is reduced by deleting "y" and "p". The list of CPU_2 observable data from M_2 cannot contain "x" anymore and the list of CPU_2 observable data from M_1 cannot contain "b" anymore. The reduced lists are as shown in the following table.

M_1			M_2			M_3		
CPU_1	CPU_2	CPU_3	CPU_1	CPU_2	CPU_3	CPU_1	CPU_2	CPU_3
a	e	c	a	e	r	c	x	z
	i	f	b	n	f			y
			s		j			s

Testing Machine Compliance to Serialization Rules

[0043] Although a processor may pre-fetch data for performance purposes, the processor is expected to discard any pre-fetched data when serialization, which is usually trigerred by the execution of certain instructions, is performed. To test if a processor CPU_b obeys the serialization rules, a combination of CPU_a fetches and the stores of CPU_b, may be used. To make a meaningful conclusion about CPU_b's compliance to the serialization rules, it must be known that the data fetched by CPU_a from M is not the last store of CPU_b into M. The exemplary serialization verification algorithm shown in FIG. 3 attempts to eliminate the stores of CPU_a that should not be observed by CPU_b, and vice versa. The algorithm takes advantage of instructions known to cause serializations. For each CPU, it is assumed that lists of all its stores into and fetches from the shared locations are available.

[0044] If CPU_a fetches data stored by CPU_b and it is determined that the data could have been overwritten by another store of CPU_b, a correlation can be made between some of the processors' streams. $Ser_{i, CPUa}$ denotes the serialization point that precedes the CPU_a fetch and $Ser_{j, CPUb}$ denotes the serialization point that follows the closest store of CPU_b that potentially could overwrite the data. When CPU_a fetches data stored by CPU_b, a cross reference point (CRP), which is a pair of serialization points $Ser_{i, CPUa}$ and $Ser_{j, CPUb}$ is created to indicate that, after Ser_i, CPU_a observed that CPU_b did not execute/reach its serialization

instruction at Ser_j. A subset of CPU_a fetches known to be stored only by CPU_b is selected from the set containing CPU_a fetches. A CRP is created for CPU_a if a store into a memory location by CPU_b is fetched by CPU_a and it is determined that there is at least another store of CPU_b that could potentially overwrite the data. For a given serialization point, at most one CRP can be defined. If a new CRP = ($Ser_{i, CPUa}$, $Ser_{j, CPUb}$) is to be created for CPU_a, a check is made to determine if CPU_a already has another CRP = ($Ser_{i, CPUa}$, $Ser_{k, CPUb}$). If one already exists, the most restrictive CRP is kept. In other words if k is less than j, the new CRP replaces the previous one.

[0045] Referring to an exemplary embodiment of an algorithm for testing machine compliance to serialization rules as depicted in FIG. 3, the terms used in FIG. 3 are defined below:

$F_{CPUi} = F_{CPUi}(M)$ = List of CPU*i* fetches from M;

St_{CPUi} = List of stores of CPU_i;

$St_{CPUi}(M)$ = List of stores of CPU_i into M;

$OW_{cpui}(M)$ = store operation of CPU_i that can overwrite the observed data from M; and

$S_{(Sern, CPUi)}$ = list of CPU_i stores that took place after Ser_n and all accumulated stores up to Ser_n.

[0046] At step 302 in FIG. 3, a loop is initiated for each pair of CPU's, referred to as each pair of CPU_a and CPU_b. The instructions labeled 304 initiate the building of CRP's for CPU_b and the checking of CPU_b fetches against CPU_a stores. This is accomplished by invoking

the instructions denoted 308 to create CRPs for CPU_b (because "y" has been set to "b"). For each CPU_a fetch (because "x" has been set to "a") of data stored by CPU_b into *M*, if other stores of CPU_b that could overwrite M did not take place, a CRP is created or updated. The instructions labeled 310 are also invoked by step 304 to check the CPU_b fetches against the CPU_a stores and the CRPs. These instructions are utilized to determine if CPU_b violated serialization rules. At step 306, the values for "b" and "y" are swapped and the reverse checking takes place: CRP's for CPU_a are built and CPU_a fetches are checked against the CPU_b stores. The algorithm depicted in FIG. 3 is performed for each pair of CPU's that utilize the shared memory location.

[0047] An example of the processing performed by the algorithm depicted in FIG. 3 is described below and assumes store and fetch instructions as shown in the following table. The numbers represent the order of each individual instruction in the stream. The "$" denotes a serialization point.

	CPU_1		CPU_2
2	$M_1 \leftarrow A_1$	4	$M_2 \leftarrow B_1$
8	$M_1 \leftarrow A_2$	7	$M_2 \leftarrow B_2$
9	$	12	$
16	$M_1 \leftarrow A_3$	20	$M_2 \leftarrow B_3$
18	$M_1 \leftarrow A_4$	22	$M_2 \leftarrow B_4$
20	$	24	$
23	$R_3 \leftarrow M_2$	25	$R_2 \leftarrow M_1$ (fetch A_2)
24	$M_1 \leftarrow A_5$	30	$M_2 \leftarrow B_5$
28	$R_1 \leftarrow M_2$ (fetch B_1)	32	$R_4 \leftarrow M_1$ (fetch A_5)
35	$M_1 \leftarrow A_6$	38	$M_2 \leftarrow B_6$
40	$	50	$

[0049] After CPU_1 observes B_1 a CRP = (20,12) is created to signify that all CPU_2 fetches after the serialization point at 12 can observe only the latest copies of all stores prior to the CPU_1 serialization at 20. As a result A_1, A_2 and A_3 cannot be observed by any fetch instruction of CPU_2 that follows in sequence the serialization instruction at 12. Any CPU_2 fetch that retrieves these data would indicate violation in serialization rules. As an example, the CPU_2 fetch at 32 is valid, as far as the serialization is concerned. On the other hand, the fetch at 25 indicates that CPU_2 is using a pre-fetched data and failed to discard it after it serialized at 12. FIG. 4 is a block diagram of an exemplary system for testing the validity of shared data in a multiprocessing system. It includes several CPUs 404 that have access to the same shared memory location 402. Any type of CPU known in the art may be utilized in an exemplary embodiment of the present invention along with any type of memory capable of being shared by

two or more processors. In an exemplary embodiment the CPUs 404 have an IBM S/390 architecture and the memory 402 is a direct access storage device (DASD). The example depicted in FIG. 4 has "n" CPUs accessing the shared memory location 404, where "n" may be any integer higher than one. In addition, each of the CPUs 404 include computer instructions to test the validity of shared data in a multiprocessing system.

[0050] An embodiment of the present invention provides for testing data block concurrency, execution sequence and serialization in multiprocessing systems. Each CPU in the multiprocessing system may store random data into shared memory locations. The method allows one or more CPUs to store identical data into the same memory location. In addition, an embodiment of the present invention tests the validity of a data fetch by a given CPU under a pipe-lines and dynamic environment. The method is applicable to any size of block concurrent update. Instruction fetching of the IBM System 390 architecture, for example, is half-word block concurrent while operands of arithmetic and logical operations are word or double-word or more (e.g., quad-word) block concurrent. Additionally, an embodiment of the present invention takes into account stores and fetches caused by arithmetic and local operations engaging in register-to-memory and memory-to-memory data transfer operations. An embodiment of the present invention also handles the case where an instruction stream and/or translation table are subject to modifications due to the actions of some of the CPUs.

[0051] As described above, the embodiments of the invention may be embodied in the form of computer-implemented processes and apparatuses for practicing those processes. Embodiments of the invention may also be embodied in the form of computer program code containing instructions embodied in tangible media, such as floppy diskettes, CD-ROMs, hard drives, or any other computer-readable storage medium, wherein, when the computer program code is loaded into and executed by a computer, the computer becomes an apparatus for

practicing the invention. An embodiment of the present invention can also be embodied in the form of computer program code, for example, whether stored in a storage medium, loaded into and/or executed by a computer, or transmitted over some transmission medium, such as over electrical wiring or cabling, through fiber optics, or via electromagnetic radiation, wherein, when the computer program code is loaded into and executed by a computer, the computer becomes an apparatus for practicing the invention. When implemented on a general-purpose microprocessor, the computer program code segments configure the microprocessor to create specific logic circuits.

[0052] While the invention has been described with reference to exemplary embodiments, it will be understood by those skilled in the art that various changes may be made and equivalents may be substituted for elements thereof without departing from the scope of the invention. In addition, many modifications may be made to adapt a particular situation or material to the teachings of the invention without departing from the essential scope thereof. Therefore, it is intended that the invention not be limited to the particular embodiment disclosed as the best mode contemplated for carrying out this invention, but that the invention will include all embodiments falling within the scope of the appended claims. Moreover, the use of the terms first, second, etc. do not denote any order or importance, but rather the terms first, second, etc. are used to distinguish one element from another.